

PyPanda: a Python Package for Gene Regulatory Network Reconstruction

David G.P. van IJzendoorn,¹ Kimberly Glass,² John Quackenbush,^{3,4,5} and Marieke L. Kuijjer^{3,4,*}

¹*Department of Pathology, Leiden University Medical Center, Leiden, the Netherlands*

²*Channing Division of Network Medicine, Department of Medicine,
Brigham and Women's Hospital, Harvard Medical School, Boston, MA, USA*

³*Department of Biostatistics and Computational Biology,
Dana-Farber Cancer Institute, Boston, MA, USA*

⁴*Department of Biostatistics, Harvard T.H. Chan School of Public Health, Boston, MA, USA*

⁵*Department of Cancer Biology, Dana-Farber Cancer Institute, Boston, MA, USA*

Summary: PANDA (Passing Attributes between Networks for Data Assimilation) is a gene regulatory network inference method that uses message-passing to integrate multiple sources of 'omics data. PANDA was originally coded in C++. In this application note we describe PyPanda, the Python version of PANDA. PyPanda runs considerably faster than the C++ version and includes additional features for network analysis. **Availability and implementation:** The open source PyPanda Python package is freely available at <http://github.com/davidvi/pypanda>. **Contact:** d.g.p.van_izendoorn@lumc.nl

I. INTRODUCTION

Accurately inferring gene regulatory networks is one of the most important challenges in the analysis of gene expression data. Although many methods have been proposed [1–4], computation time has been a significant limiting factor in their widespread use. PANDA (Passing Attributes between Networks for Data Assimilation) is a gene regulatory network inference method that uses message passing between multiple 'omics data types to infer the network of interactions most consistent with the underlying data [5]. PANDA has been applied to understand transcriptional programs in a variety of systems [6–8].

Here we introduce PyPanda, a Python implementation of the PANDA algorithm, following the approach taken in Glass *et al.* [9] and optimized for matrix operations using NumPy [10]. This approach enables the use of fast matrix multiplications using the BLAS and LAPACK functions, thereby significantly decreasing runtime for network prediction compared with the original implementation of PANDA, which was coded in C++ and used for-loops [9]. We observe further speed increase over the C++-code because PyPanda automatically uses multiple processor-cores through the NumPy library. We have also expanded PyPanda to include common downstream analyses of PANDA networks, including the calculation of network in- and out-degrees and the estimation of single-sample networks using the recently developed LIONESS algorithm [11].

II. APPROACH

II.1. Comparing PANDA C++-code to Python-code

We compared the C++-code and Python-code versions of PANDA using several metrics. First, we assessed the two implementations by comparing the number of lines of code. Using the *cloc* utility we counted the number of lines of C++-code and Python-code. The C++-code counted 1132 lines of code. The Python-code counted 258 lines of code, significantly shorter (4.4 times) than the C++-code. The Python-code also includes features such as the LIONESS equation and in- and out-degree calculation. Without these features the Python-code is only 155 lines of code. Because the Python implementation is much more concise than the C++-code it is easier to interpret and modify.

Next we performed a speed comparison test between the C++-code and the Python-code. We used built-in timing functions for both languages, directly before and after the message passing part of the code as this is the step that consumes the most time [9]. For the C++-code, we used *gettimeofday()* to record time in milliseconds before and after the message passing algorithm. For the Python code we implemented the *time.time()* function around the message passing algorithm. The C++-code was compiled using the *clang* compiler (version 3.8.0) with speed optimization flag -O3. Python (version 2.7.10) was used with NumPy (version 1.10.1) using the BLAS and LAPACK algebraic functions. All analyses were run on a server running x86_64 GNU/Linux.

The speed of the network prediction was tested using simulated networks of $Ne = Na$ dimensions, where Ne is the number of effector nodes and Na is the number of affected nodes [9]. For each of several different network sizes ($Ne = Na = 125$ to $Ne = Na = 2000$ nodes, in steps of 125) we generated ten random 'motif data' networks according to the method described in Glass *et*

* mkuijjer@jimmy.harvard.edu

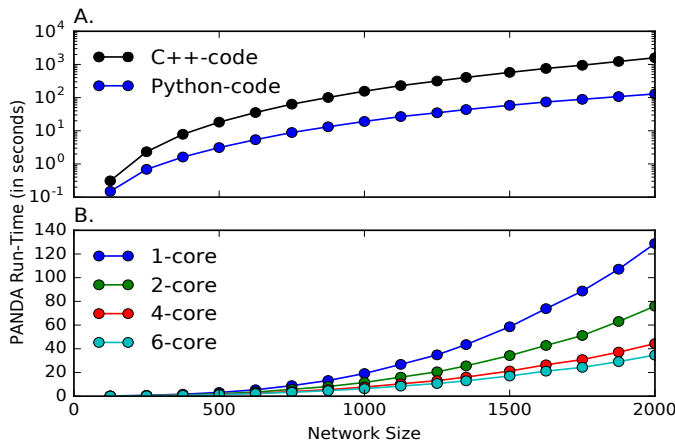


Figure 1. Speed comparison for network reconstruction on networks of different sizes using (A) the C++-code and the Python-code, (B) the Python-code running on a single CPU compared with multicore (6 CPU cores).

al. [9]. We then ran the Python and C++ versions of PANDA using these simulated motif data together with identity matrices for the protein-protein interaction and co-expression information. For runs on the same initial ‘motif data’ networks, we verified that the C++-code and Python-code returned exactly the same output network, as expected due to the deterministic nature of PANDA.

The C++-code only uses one CPU core. In comparing the C++-code with the Python-code using a single core, we found a 2.07-fold speed-up relative to the C++-code for the smallest network ($N_e = N_a = 125$) tested. The speed increase of the Python-code over the C++-code became larger as the network size increased. For example, the Python-code performed 12.31 times faster for the largest network ($N_e = N_a = 2000$) (Figure 1A). Recorded run times across the ten random networks had a standard deviation of 0.04s and 2.59s for the smallest ($N_e = N_a = 125$) and largest ($N_e = N_a = 2000$) networks, respectively using the C++ code. Using the Python code these were reduced to 0.03s and 0.099s.

Given the abundance of multicore computing resources currently available, we also tested the speed increase when running the Python-code on multiple cores compared with running the Python-code on a single core. We found that for the smallest network the speed was 1.45 times faster when using 6 cores compared with using only a single core; for the largest network the speed increase was 3.7-fold (Figure 1B).

This increase in speed enables reconstruction of networks with larger numbers of regulators and target genes. For example, using the Python-code significantly decreases the time required to infer a human gene regulatory network ($N_e = 1000$, $N_a = 20000$), from approximately 18h with the C++-code to only about 2h with the Python-code. This speed-up is especially important as transcription factor motif databases are frequently updated to include more motifs. Further, the decreased run-

ning time helps to enable the estimation of network significance by making the use of bootstrapping/jackknifing methods much more feasible.

II.2. Additional Features

In addition to reconstructing one regulatory network based on a data set consisting of multiple samples, PyPanda can also reconstruct single-sample networks using the LIONESS algorithm [11]. In PyPanda, the LIONESS method uses PANDA to infer an ‘aggregate’ network representing a set of N input samples, infers a network for $N - 1$ samples, and then applies a linear equation to estimate the network for the sample that had been removed. The process is then repeated for each sample in the original set, producing N single-sample networks. PyPanda can also use LIONESS to reconstruct single-sample networks based on Pearson correlation.

PyPanda also includes functions to calculate in-degrees (the sum of edge weights targeting a specific gene) and out-degrees (the sum of edge weights pointing out from a regulator to its target genes). These summary metrics can be used for downstream network analysis [6].

III. CONCLUSION

PANDA is a proven method for gene regulatory network inference but, like most sophisticated network inference methods, its runtime has limited its utility. The Python implementation of PANDA uses matrix operations and incorporates the NumPy libraries, resulting in a significant simplification of the code and a dramatic increase in computing speed, even on a single processor. When applied to a test data set and run on multiple processing cores, this increase in speed was even greater, decreasing processing times by a factor of 46 relative to the original C++-code. This creates opportunities to greatly expand the use of PANDA and to implement additional measures of network significance based on bootstrapping/jackknifing. PyPanda also includes the LIONESS method, which allows inference of single-sample networks, as well as a number of other useful network metric measures. The open source PyPanda package is freely available at <http://github.com/davidvi/pypanda>.

ACKNOWLEDGMENTS

The authors would like to thank Judith V.M.G. Bovée, MD, PhD and Karoly Suzhai, MD, PhD for thoughtful discussions and Cho-Yi Chen, PhD for testing PyPanda.

FUNDING

HL105339 to J.Q., K.G., M.L.K.] and Leiden University Fund [5259/4-6-2015/Gg to D.G.P.IJ].

This work has been supported by the National Institutes of Health [R01 HL111759 to J.Q., K.G., P01

-
- [1] G. Altay, M. Asim, F. Markowetz, and D. E. Neal, “Differential c3net reveals disease networks of direct physical interactions,” *BMC Bioinformatics* **12**, 296 (2011).
 - [2] J. J. Faith, B. Hayete, J. T. Thaden, I. Mogno, J. Wierzbowski, G. Cottarel, S. Kasif, J. J. Collins, and T. S. Gardner, “Large-scale mapping and validation of *escherichia coli* transcriptional regulation from a compendium of expression profiles,” *PLOS Biol.* **5**, e8 (2007).
 - [3] K. Lemmens, T. Dhollander, T. De Bie, P. Monsieurs, K. Engelen, B. Smets, J. Winderickx, B. De Moor, and K. Marchal, “Inferring transcriptional modules from chip-chip, motif and microarray data,” *Genome Biol.* **7**, R37 (2006).
 - [4] J. Ernst, Q. K. Beg, K. A. Kay, G. Balázs, Z. N. Oltvai, and Z. Bar-Joseph, “A semi-supervised method for predicting transcription factor–gene interactions in *escherichia coli*,” *PLOS Comput. Biol.* **4**, e1000044 (2008).
 - [5] K. Glass, C. Huttenhower, J. Quackenbush, and G.-C. Yuan, “Passing messages between biological networks to refine predicted interactions,” *PLOS ONE* **8**, e64832 (2013).
 - [6] K. Glass, J. Quackenbush, E. K. Silverman, B. Celli, S. I. Rennard, G.-C. Yuan, and D. L. DeMeo, “Sexually-dimorphic targeting of functionally-related genes in *copd*,” *BMC Syst. Biol.* **8**, 118 (2014).
 - [7] K. Glass, J. Quackenbush, D. Spentzos, B. Haibe-Kains, and G.-C. Yuan, “A network model for angiogenesis in ovarian cancer,” *BMC Bioinformatics* **16**, 115 (2015).
 - [8] T. Lao, K. Glass, W. Qiu, F. Polverino, K. Gupta, J. Morrow, J. D. Mancini, L. Vuong, M. A. Perrella, C. P. Hersh, et al., “Haploinsufficiency of hedgehog interacting protein causes increased emphysema induced by cigarette smoke through network rewiring,” *Genome Med.* **7**, 12 (2015).
 - [9] K. Glass, J. Quackenbush, and J. Kepner, “High performance computing of gene regulatory networks using a message-passing model,” *High Performance Extreme Computing Conference (HPEC)*, IEEE (2015).
 - [10] S. Van Der Walt, S. C. Colbert, and G. Varoquaux, “The numpy array: a structure for efficient numerical computation,” *Comput. Sci. & Eng.* **13**, 22 (2011).
 - [11] M. L. Kuijjer, M. Tung, G. Yuan, J. Quackenbush, and K. Glass, “Estimating sample-specific regulatory networks,” *arXiv preprint:1505.06440* (2015).